Assignment 2 DAA — Answers

Ivan Aristy — iae225

1. Question 1: Hashing

1.1. Problem A

.

(a) 5 points Consider a hash table implemented as an array, $T[]$, indexed from $0 \dots 16$. Hashing is carried out using double hashing, with $h_1(k) = k \mod 17$ and $h_2(k) = k^2 + 1 \mod 17$. Show the insertion of: $45, 99, 32, 96, 25, 36, 83, 27, 21, 49, 51, 8$, or explain why it is that certain keys are not inserted.

```
index
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
h1(k) = k mod 17
h2(k) = k^2 + 1 mod 17

45
h1(45) = 11
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
[ , , , , , , , , , , ,45,  ,  ,  ,  ,  ]

99
h1(99) = 14
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
[ , , , , , , , , , , ,45,  ,  ,99,  ,  ]

32
h1(32) = 15
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
[ , , , , , , , , , , ,45,  ,  ,99,32,  ]

96
h1(96) = 11
h2(96) = 3 = 14
h2(96) = 3 = 17 -> 0
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
[96, , , , , , , , , , ,45,  ,  ,99,32,  ]

25
h1(25) = 8
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[96,  ,  ,  ,  ,  ,  ,  , 25,  ,  ,45,  ,  ,99,32,  ]

36
h1(36) = 2
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[96,  , 36,  ,  ,  ,  ,  , 25,  ,  ,45,  ,  ,99,32,  ]

83
h1(83) = 15
h2(83) = 5
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[96,  ,36,83,  ,  ,  ,  ,25,  ,  ,45,  ,  ,99,32,  ]

27
h1(27) = 10
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[96,  ,36,83,  ,  ,  ,  ,25,  ,27,45,  ,  ,99,32,  ]

21
h1(21) =
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[96,  ,36,83,21,  ,  ,  ,25,  ,27,45,  ,  ,99,32,  ]

49
h1(49) = 15
h2(49) = 5 x 3
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[96,  ,36,83,21,  ,  ,  ,25,  ,27,45,  ,49,99,32,  ]

51
h1(51) = 0
h2(51) = 1
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[96,51,36,83,21,  ,  ,  ,25,  ,27,45,  ,49,99,32,  ]

8
h1(8) = 8
h2(8) = 14
```

```
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[96,51,36,83,21, 8,  ,  ,25,  ,27,45,  ,49,99,32,  ]
```

1.1. Problem B

(b) 5 points Repeat the above, but where hashing is carried out using the quadratic hash function $h(k,i) = k + 2i + 3i^2 \mod 17$. Show the insertion of the keys from part (a), or explain why it is that certain keys are not inserted.

```
h(k,i) = k + 2i + 3i^2 mod 17

Explain why it is that certain keys are not inserted

45
h(45,0) = k + 2i + 3i^2 mod 17 = 11
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,45,  ,  ,  ,  ,  ]

99
h(99,0) = k + 2i + 3i^2 mod 17 = 14
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,45,  ,  ,99,  ,  ]

32
h(32,0) = k + 2i + 3i^2 mod 17 = 15
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,45,  ,  ,99,32,  ]

96
h(96,0) = k + 2i + 3i^2 mod 17 = 11
h(96,1) = k + 2i + 3i^2 mod 17 = 16
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,45,  ,  ,99,32,96]

25
h(25,0) = k + 2i + 3i^2 mod 17 = 8
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[  ,  ,  ,  ,  ,  ,  ,  ,25,  ,  ,45,  ,  ,99,32,96]

25
h(25,0) = k + 2i + 3i^2 mod 17 = 8
h(25,1) = k + 2i + 3i^2 mod 17 = 13
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[  ,  ,  ,  ,  ,  ,  ,  ,25,  ,  ,45,  ,25,99,32,96]

36
h(36,0) = k + 2i + 3i^2 mod 17 = 2
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[  ,  ,36,  ,  ,  ,  ,  ,25,  ,  ,45,  ,25,99,32,96]

83
h(83,0) = k + 2i + 3i^2 mod 17 = 15
h(83,1) = k + 2i + 3i^2 mod 17 = 3
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[  ,  ,36,83,  ,  ,  ,  ,25,  ,  ,45,  ,25,99,32,96]

27
h(27,0) = k + 2i + 3i^2 mod 17 = 10
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[  ,  ,36,83,  ,  ,  ,  ,25,  ,27,45,  ,25,99,32,96]

21
h(21,0) = k + 2i + 3i^2 mod 17 = 10
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[  ,  ,36,83,21,  ,  ,  ,25,  ,27,45,  ,25,99,32,96]

49
h(49,0) = k + 2i + 3i^2 mod 17 = 15
h(49,1) = k + 2i + 3i^2 mod 17 = 3
h(49,2) = k + 2i + 3i^2 mod 17 = 14
h(49,3) = k + 2i + 3i^2 mod 17 = 14
h(49,4) = k + 2i + 3i^2 mod 17 = 3
h(49,5) = k + 2i + 3i^2 mod 17 = 15
h(49,6) = k + 2i + 3i^2 mod 17 = 15
h(49,7) = k + 2i + 3i^2 mod 17 = 6 we reach it here but we could've
broken out of the cycle before if i was bound to 6.
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[  ,  ,36,83,21,  ,49,  ,25,  ,27,45,  ,25,99,32,96]
```

```
51
h(49,0) = k + 2i + 3i^2 mod 17 = 15
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[51,  ,36,83,21,  ,49,  ,25,  ,27,45,  ,25,99,32,96]

8
h(49,0) = k + 2i + 3i^2 mod 17 = 15
h(49,1) = k + 2i + 3i^2 mod 17 = 13
h(49,2) = k + 2i + 3i^2 mod 17 = 7
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[51,  ,36,83,21,  ,49, 8,25,  ,27,45,  ,25,99,32,96]
```

1.1. Problem C

(c) 3 points Repeat the hashing once again, this time using function $h(k) = k^2 \mod 17$, where collisions are resolved with chaining.

```
[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[51,  ,45,  ,32,  ,  ,  ,  ,99,  ,  ,  ,25,  ,27,21]
[  ,  ,96,  ,36,  ,  ,  ,  ,  ,  ,  ,  , 8,  ,  ,  ]
[  ,  ,  ,  ,83,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ]
[  ,  ,  ,  ,49,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ]
[  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ]
[  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ]
[  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ]
```

1.1. Problem D

(d) 3 points Of the above three hash tables, which method requires the most probes in order to **insert** key 15 in the table?

The hash table in 1b requires the most probes.

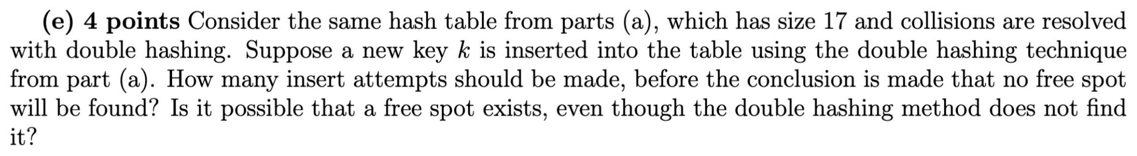
```
h(k,i) = k + 2i + 3i^2 mod 17

h1(k) = k mod 17
h2(k) = k^2 + 1 mod 17

[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[51,  ,36,83,21,  ,49, 8,25,  ,27,45,  ,25,99,32,96]
h(15,0) = k + 2i + 3i^2 mod 17 = 15
h(15,1) = k + 2i + 3i^2 mod 17 = 3
h(15,2) = k + 2i + 3i^2 mod 17 = 14
h(15,3) = k + 2i + 3i^2 mod 17 = 14
h(15,4) = k + 2i + 3i^2 mod 17 = 3
h(15,5) = k + 2i + 3i^2 mod 17 = 15
h(15,6) = k + 2i + 3i^2 mod 17 = 16
h(15,7) = k + 2i + 3i^2 mod 17 = 6

[0 , 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15,16]
[96,51,36,83,21, 8,  ,  ,25,  ,27,45,  ,49,99,32,  ]
h1(15) = k mod 17 = 15
h2(15) = k^2 + 1 mod 17 = 5
h2(15) = k^2 + 1 mod 17 = 5
h2(15) = k^2 + 1 mod 17 = 5
h2(15) = k^2 + 1 mod 17 = 5
```

1.1. Problem E



(e) 4 points Consider the same hash table from parts (a), which has size 17 and collisions are resolved with double hashing. Suppose a new key k is inserted into the table using the double hashing technique from part (a). How many insert attempts should be made, before the conclusion is made that no free spot will be found? Is it possible that a free spot exists, even though the double hashing method does not find it?

We should do 17 attempts because 17 is prime and our table size is also 17, which... is prime. For bad double hashing, we'd need a secondary hash function with a step size that leads to cycling, but, in our case, due to the prime table size, and the mod 17 with a +1 inside the function, we can be fairly certain that we're ok.

2. Question 2: Selection

2.1. Problem A

.

(a) 6 points Carry out the *Select* algorithm on set below using $k = 14$ (return the element of rank 14). Show your steps! You do not need to show the steps of the partition algorithm. Instead, ensure that the output of the partition algorithm maintains the elements in their original relative order. You must carry out *all* recursive calls to the Select algorithm. You may assume that the base case of the Select algorithm is any input of at most 5 elements, in which case it returns the element of rank k.

**When taking the median of an even number of elements, use the lower median by default*

$$25, 37, 52, 14, 89, 35, 83, 53, 31, 86, 99, 46, 66, 34, 22, 2, 8, 90, 30, 68,$$

$$21, 17, 84, 29, 77, 45, 33, 41, 19, 53, 42, 93, 23, 18, 91$$

```
array pablo = 25, 37, 52, 14, 89, 35, 83, 53, 31, 86, 99, 46, 66, 34,
22, 2, 8, 90, 30, 68, 21, 17, 84, 29, 77, 45, 33, 41, 19, 53, 42, 93,
23, 18, 91

Select(k = 14, Arr = pablo)

Step 1:
25, 37, 52, 14, 89,
35, 83, 53, 31, 86,
99, 46, 66, 34, 22,
 2,  8, 90, 30, 68,
21, 17, 84, 29, 77,
45, 33, 41, 19, 53,
42, 93, 23, 18, 91,

Step 2: Sort
14, 25, 37, 52, 89,
31, 35, 53, 83, 86,
22, 34, 46, 66, 99,
 2,  8, 30, 68, 90,
17, 21, 29, 77, 84,
19, 33, 41, 45, 53,
18, 23, 42, 91, 93,

Step 3: Median
17, 21, 29, 77, 84,
 2,  8, 30, 68, 90,
14, 25, 37, 52, 89,
19, 33, 41, 45, 53,
18, 23, 42, 91, 93,
22, 34, 46, 66, 99,
31, 35, 53, 83, 86,

41

Step 4: rank median
17, 21, 29, 77, 84, 3 2
 2,  8, 30, 68, 90, 3 2
14, 25, 37, 52, 89, 3 2
19, 33, 41, 45, 53, 2 2
18, 23, 42, 91, 93, 2 3
22, 34, 46, 66, 99, 2 3
31, 35, 53, 83, 86, 2 33

= 18th

Step 5: recursive call

Select(k = 14, Arr = pablo - the bigger elements)

17, 21, 29,  2,  8, 30, 14, 25, 37, 19, 33, 18, 23, 22, 34, 31, 35,

Step 1: Divide
17, 21, 29,  2,  8,
30, 14, 25, 37, 19,
33, 18, 23, 22, 34,
31, 35,

Step 2: Sort
 2,  8, 17, 21, 29,
14, 19, 25, 30, 37,
18, 22, 23, 33, 34,
31, 35,

Step 3: Median of medians
 2,  8, 17, 21, 29,
18, 22, 23, 33, 34,
14, 19, 25, 30, 37,
31, 35,
23

Step 4: rank median
 2,  8, 17, 21, 29, 4 1
18, 22, 23, 33, 34, 2 2
14, 19, 25, 30, 37, 2 1
31, 35,             0 2

= 9th

Step 5: k − r, call on bigger, new r = 5

21, 29, 33, 34, 25, 30, 37, 31, 35

Step 1:
21, 29, 33, 34, 25,
30, 37, 31, 35,

Step 2:
21, 25, 29, 33, 34,
30, 31, 35, 37,

Step 3:
21, 25, 29, 33, 34,
30, 31, 35, 37,

Median = 29

Step 4:
21, 25, 29, 33, 34, 2 2
30, 31, 35, 37,     0 4

= 3rd

Step 5: r < k , call bigger, new r = 2

Step 1:
29, 33, 34, 30, 31,
35, 37,

Step 2:
29, 30, 31, 33, 34,
35, 37,

Step 3:
29, 30, 31, 33, 34,
35, 37,

= 31

Step 4:
29, 30, 31, 33, 34, 2 2
35, 37,             0 2

3rd

Step 5: r > k, call smaller

Base case reached, we only have 29 and 30 left. Return 30.
```

2.1. Problem B

$$25, 37, 52, 14, 89, 35, 83, 53, 31, 86, 99, 46, 66, 34, 22, 2, 8, 90, 30, 68,$$

$$21, 17, 84, 29, 77, 45, 33, 41, 19, 53, 42, 93, 23, 18, 91$$

(b) 6 points Repeat the above, using the Randomized-select algorithm. Seeing as you don't have a random-number generator on hand, you will select your "random" pivot with bias: always select the *first* element in your array as your random pivot. As above, you don't need to show the execution of the partition algorithm, but ensure for consistency that the output of the partition algorithm maintains the elements in their original relative order.

```
array pablo = 25, 37, 52, 14, 89, 35, 83, 53, 31, 86, 99, 46, 66, 34,
22, 2, 8, 90, 30, 68, 21, 17, 84, 29, 77, 45, 33, 41, 19, 53, 42, 93,
23, 18, 91

RandomizedSelect(k = 14, Arr = pablo)

Recur 1: 25 k = 14
Smaller = [14, 22, 2, 8, 21, 17, 19, 23, 18]
Bigger = [37, 52, 89, 35, 83, 53, 31, 86, 99, 46, 66, 34, 90, 30, 68,
84, 29, 77, 45, 33, 41, 53, 42, 93, 91]

Ranks is 10,
r < k, go on bigger, offset to 4

Recur 2: 37 k = 4
[37, 52, 89, 35, 83, 53, 31, 86, 99, 46, 66, 34, 90, 30, 68, 84, 29,
77, 45, 33, 41, 53, 42, 93, 91]
Smaller = [35, 31, 34, 30, 29, 33]
Bigger = [52, 89, 83, 53, 86, 99, 45, 66, 90, 68, 84, 77, 45, 41, ....]

Rank is 7
r > k, go on smaller

Recur 3: 35 k = 4
[35, 31, 34, 30, 29, 33]
Smaller = [31, 34, 30, 29, 33]
Bigger =
r > k, go on smaller

Recur 4: 31 k = 4
[31, 34, 30, 29, 33]
Smaller = [29,30]
Bigger = [33, 34]
r = 3
r < k, go on bigger, k - r = 1

Recur 5: k
2 elements left, return 33... Looks like we made a mistake on Problem
A...
```

2.1. Problem C

$$\text{Step 1: Divide n elements into n groups of 6} = \mathcal{O}(n)$$
$$\text{Step 2: Sort groups of size 6 } = \mathcal{O}(n)$$
$$\text{insertion sort, } c \times \frac{n}{6}$$
$$\text{Step 3: Find medians} = T\left(\frac{n}{6}\right)$$
$$\text{Step 4: Partition} = \mathcal{O}(n)$$
$$\text{Step 5 Recursive call through rank}$$
$$\text{When k < r, on the green section we at most go through } \frac{1}{2} \times \frac{n}{6} + \frac{1}{2} \times \frac{3n}{6} \geq \frac{4n}{12} = \frac{1n}{3}$$
$$\therefore \text{larger elements are } T\left(\frac{2n}{3}\right)$$

$$T(n) = T\left(\frac{n}{6}\right) + T\left(\frac{2n}{3}\right) + \mathcal{O}(n)$$
$$T(n) = T\left(\frac{n}{6}\right) + T\left(\frac{2n}{3}\right) + dn$$
$$\left(\frac{n}{6}\right) + T\left(\frac{2n}{3}\right) + dn \leq cn$$
$$\text{Inductive Hypothesis:}$$
$$T\left(\frac{n}{6}\right) = \frac{cn}{6} \wedge T\left(\frac{2n}{3}\right) = \frac{c2n}{3}$$
$$T(n) = T\left(\frac{n}{6}\right) + T\left(\frac{2n}{3}\right) + dn \leq \frac{cn}{6} + \frac{c2n}{3} + dn$$
$$\leq \frac{cn}{6} + \frac{c2n}{3} + dn$$
$$\leq \frac{cn}{6} + \frac{c4n}{6} + dn$$
$$\leq \frac{c5n}{6} + dn$$
$$\leq \frac{c5n}{6} + dn$$
$$\leq n \times (\frac{5c}{6} + dn)$$
$$\text{So this is only true when c is >= to 6d}$$

3. Question 3: Heaps

3.1. Problem A

.

(a) 10 points Below are three different methods for building a max-heap. Suppose that $A[1, 2, \ldots, n]$ initially contains n elements, which are not necessarily in heap format. You may assume that the value of $A.heapsize = n$. For each method, justify whether the method correctly. builds a max-heap. If so, justify your answer. If not, provide an example showing it doesn't work.

```
METHOD 1:
Initial call: Make-Heap1($A, n$)
Make-Heap1($A, i$)
    If $i > 1$
        Bubble-up($A$,$i$)
        Make-Heap1($A$,$i - 1$)
```

```
METHOD 2:
Initial call: Make-Heap2($A, n$)
Make-Heap2($A, i$)
    If $i \geq 1$
        Bubble-down($A$,$i$)
        Make-Heap2($A$,$i - 1$)
```

```
METHOD 3:
Initial call: Make-Heap3($A, 1$)
Make-Heap3($A, i$)
    If $i \leq A.heapsize$
        Bubble-up($A$,$i$)
        Make-Heap3($A$,$i + 1$)
```

Method 1:
We call on an array with a heap size of n
Make heap first calls with size of n
Bubbles up the last value
And It then progressively calls itself on a size 1 less than itself
Since calling bubble up on a valid heap doesn't break heap properties,
this method will work.

Method 2:
We call on the last element like before
We bubble down on every element
since the input array is sorted,
bubble down can perform properly
if it wasn't we'd get issues,
but by sorting we guarantee that the lowest values are bubbled down to their correct positions at the end of execution

Method 3:
This method will work perfectly,
since we can just think of this as bubbling up elements
that are continuously inserted into a new array.
they do happen to be sorted here,
but this is just regular bubble up insertion.

3.1. Problem B

(b) 8 points A *min-max-heap* is a complete binary tree containing alternating *min* and *max* levels. The root node is the **minimum** of all nodes in the tree. The nodes at the next level are the largest nodes of their subtrees. For each level after that, a node that is on a min level is the minimum of all nodes in its subtree, and a node that is on a max level, is the maximum of all nodes in its subtree. Below is an example of a min-max tree. Assume that the elements are stored in array $A[]$ as for usual heaps, where $A.heapsize$ indicates the size of the heap. Your job is write the pseudo-code for a procedure that **inserts** into this type of heap. Call your procedure MinMaxInsert(A, k), which inserts k into the heap, and carries out the necessary swaps in order to maintain the min-max heap property.

```
MinMaxInsert(A, k):
    // Insert at the top
    A[A.heapsize] = k
    A.heapsize++;

MinInsert():

MaxInsert():

getgrandparent(A, k)
```

3.1. Problem C

(c) 6 points Suppose A is a sorted list of exactly n elements, sorted in decreasing order. Suppose B a max-heap of exactly n elements. Two students are given the task of creating a sorted list of the combined elements from set A and B. Below are their approaches:

Student 1: Add all elements of A and B to a new array C of size $2n$. Then uses bottom-up heap building, resulting in a max heap in array C. Next, run HeapSort on array C.

Student 2: Repeat until A and B are empty: Compare first element of A with element $B[1]$. If top element of A is larger, extract it to the output list. If $B[1]$ is larger, call Delete-max, and store to the output list. If at some point B is empty, simply concatenate all remaining elements of A to the output list. If at some point A is empty, continue calling Delete-max on B until it is empty.

Determine the worst-case runtime for each method above. Is there one student with a more efficient approach?

Student 1:
Adding all elements to the new array is just O(n).
Building the heap using bottom up method is O(n) as well.
Heapsort is the bottleneck at O(nlogn)
Complexity is O(nlogn)

Student 2:
Comparisons run in O(1)
We do O(n) comparisons for all the elements in A and B.
removing from A is O(1)
delete max is O(logn)
Assume A is empty and B is not,
our bottleneck is O(nlogn) on the worst case.

Asymptotically, both have the same runtime complexity; however, student 2 is technically correct, which is the best kind of correct. Student 2's approach is more efficient since it only touches elements once, preventing lots of O(n) operations which are hidden by the

O(nlogn) upper bound. Additionally, on the best case, where A is full and B is empty, student 2 will run the algo in O(n)

4. Question 4: Lower Bounds and Linear time Sorting

4.1. Problem A

Option 1:
Select will run in O(n)
Storing items will take O(n)
At this point we have sqrt(n) items
Quicksorting will take O(sqrt(n) log(sqrt(n))),
which, unless im missing some algebra magic,
is still O(nlogn)

Option 2:
This depends...
It will run in O(n) due to using bucketsort,
but we are not necessarily guaranteed that our first bucket will only contain 10 elements.
Sure, we say it's an uniform distribution, but I doubt that we can take it for granted that bucket 1 will always hold 10 elements...

Option 3:
This doesn't work.
We said "Let A[] be an array of n real (decimal) numbers"
Counting sort doesn't work on the reals.
It would be Theta of (infinity)

4.1. Problem B

4.1. Problem C

5. Notes for self